9/4/01, SDC

WEST

End of Result Set

Generate Collection | Print

L15: Entry 24 of 24 File: TDBD Mar 1, 1995

TDB-ACC-NO: NN9503477

DISCLOSURE TITLE: Referential Integrity Implementation Details and Advantages

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, March 1995, US

VOLUME NUMBER: 38
ISSUE NUMBER: 3
PAGE NUMBER: 477 - 488

PUBLICATION-DATE: March 1, 1995 (19950301)

CROSS REFERENCE: 0018-8689-38-3-477

DISCLOSURE TEXT:

This document contains drawings, formulas, and/or symbols that will not appear on line. Request hardcopy from ITIRC for complete article. A method for providing referential integrity within a database system that provides both superior functionality and ease-of-use is disclosed. This solution also allows for easy referential constraint management without drastically impacting system storage requirements as other methods have done. In addition, this method is also standards compatible with another key database feature, triggers, and can be applied to distributed database systems as well. An important function within any relational data base system is that of referential integrity. Referential Integrity (RI) ensures data consistency between related columns of two different tables or the same table. The defined and enforced relationship between these related columns is known as a referential constraint. A referential constraint is an assertion that non-null values of a "foreign key" are valid only if they also appear as values of the "primary" key in the related table. To guarantee a data base's referential integrity, the data base system must ensure that non-null foreign key values have a matching primary key value. Although the concept of referential integrity is not new, the RI design disclosed here is unique and advantageous in many aspects. Disclosed are four methods for referential integrity. 1. Constraint Management - Ease of use in managing table constraints with missing pieces: The user can have tables with partial constraints. To define a constraint relationship, the parent table and dependent table must both exist. The system allows the user to define a constraint relationship even when the object (a file member in DB2/400*) that contains the data does not yet exist. The mechanism to support this is the system cross reference file. When a constraint is defined, an entry for the constraint will be registered in the cross reference file. Once the parent and dependent table are associated with data objects (members), a dependent access path (index) will be created for the constraint definition, the constraint will be registered in the cross reference file, and the constraint will be placed in the established state. Other systems also use indexes for RI constraint definitions. However, most of those systems always create their "own" index for constraint enforcement. This method has the database first look and see if the table has an existing index that can be shared for constraint enforcement before creating an index. This provides the end user with a couple of advantages. First, additional storage (DASD) for a separate index is not wasted by the database system in the case where this method is able to share an existing index (which should be fairly common).

Second, changes to that table will not suffer the performance degradation associated with maintaining "extra" indexes over a table. The system will also automatically attempt to reestablish the constraint relationships when a table (or file) moves to another database (or library) via a copy operation, table rename, or load operation from media. The system cross reference file is again used to determine which tables and columns participate in a relationship so that the system can redefine or reestablish the constraint relationship when a table is "moved" by the user. 2. Constraint Enforcement (& Trigger Interaction) - Disclosed is a method for ensuring that the enforcement of table constraints and the execution of trigger programs is consistent and predictable.

This method relies on deferring constraint checking until the end of the statement or transaction is necessary to prevent any table/constraint restrictions and to prohibit table constraints and trigger operations from affecting Structured Query Language (SQL) selection results as dictated by the American National Standards Institute (ANSI) SQL-92 standards. - An enforcement ordering algorithm and a deferred enforcement mode to ensure that the interaction between table constraints and trigger operations is consistent and predictable. Some definitions of terms used in this publication will first be discussed. o Statement - One or more row manipulations due to a single user request. For this paper, a statement is only defined within commitment control. o Transaction - One or more statements associated with a commitment control boundary.

o Enforcement Cycle - The mechanism for defining the boundaries of either a statement or transaction. Also known as a nested referential constraint transaction. The execution of a statement by a database user results in the manipulation of a row or rows by the Data Base Management System (DBMS). These row manipulations can be the result of enforcing the table constraints as a result of the user request.

For example, referential update cascade rule enforcement may cause dependent records to be updated or trigger programs may cause parent records to be inserted. The ANSI SQL-92 standards require the evaluation of table constraints to be performed at the end of the statement or transaction. This evaluation deferral must be isolated from the primary transaction. For example, if a user has started a primary commit cycle and has performed a number of updates before issuing a delete causing a delete cascade to occur, an exception causing the delete cascade to get rolled back should not roll back the users original updates. Evaluation of the constraints is then performed at the end of this enforcement cycle to meet the standards. Thus, the evaluation of constraints is performed after all user initiated row manipulation within the statement have completed, including manipulations caused by trigger programs. The evaluation is done in the following order to eliminate the need to restrict environments of cyclic or connected tables and to consistently and predictably enforce table constraints on files that are trigger enabled. See Fig. 1 for an example of delete connected files. a. Start enforcement cycle b. Before Trigger Program execution c. Restrict RI rules are enforced first - No Action rules differ from Restrict rules in that No action rules are enforced last. Since they are enforced first, Restrict rule enforcement cannot be influenced by other constraints or trigger operations. d. After Trigger Program Execution. e. End Enforcement Cycle - The end of the enforcement cycle indicates that table constraint enforcement can now begin. Notice that RI enforcement for all rules other than Restrict will be performed on the after affects of the trigger operations.

f. All Delete Cascade rules are enforced next - This allows the rows deleted as a result of the Delete Cascade rule to not interfere with the enforcement of the remaining rules. g. Update Cascade and Delete/Update Set Null and Set Default rules are enforced next. h. No Action rule enforcement is performed - This enforcement is performed against the results of all other rule enforcement that caused row manipulations. This allows the previous rule row manipulations the chance to clean up any RI violation that otherwise would have caused this transaction to fail. i. Enforcement of unique table constraints is performed last - Duplicate key exception cannot be raised until the end of a statement. Again, this allows the previously enforced rules the opportunity to clean up this duplicate situation. This rule enforcement ordering provides the functionality of allowing table constraint enforcement to be performed on the after images of the trigger operations. This would, for example, allow an insert trigger program to be defined over a parent file

that would insert the matching dependent record. - This could be a users application for populating his files having referential constraints. If the record in table T1 is deleted, the order in which the constraints is enforced determines the results. o If the delete noaction is enforced first, the operation results in a constraint violation since a dependent record of 1 exists in table T3.

o If the delete cascades to T2 is enforced first followed by the delete cascade to T3 followed by the enforcement of the delete noaction, then the noaction rule does not result in a violation since the cascade from table T2 has deleted the dependent row in T3. Since enforcement is delayed until the end of the enforcement cycle, selection results are not impacted by the enforcement during the user statement are not affected. In our PartID example, if all rows with a PartID='99' were selected for deletion, an update cascade rule adding a new Partid='99' would not occur until the selection of the original value was complete. Since the enforcement ordering algorithms ensures that all results are predictable for all constraint environments, there is no need for restrictions. In the delete connected tables example of Fig. 1, since our rule ordering dictates that cascade rules are handled before no action rules, the result is that all rows are deleted. This result is consistent and predictable knowing the order of rule enforcement. When the user is using statement level constraint deferment, the enforcement cycle is defined to be a nested cycle within the users primary commit cycle. Thus, any exceptions taken during constraint enforcement results in the rollback of the nested cycle, leaving the users primary cycle unaffected. - The rule ordering algorithm implemented in this method defines that delete cascade rules are always enforced before no action rules. Thus, it is predictable and consistent that the result of this operation is that all three tables become empty after the delete of the record in table T1. 3. Distributed Referential Integrity - This methods provides a means for providing RI enforcement across multiple remote server sites when processing distributed transactions.

The key to this method is extending the "local" DB2/400 RI solution with some help from the distributed a distributed database network with some help from a transaction manager. This distributed solution provides real-time referential constraint enforcement and full recovery for any referential transaction; these two properties are a must for ensuring that the data integrity and data consistency exist at ALL sites in a distributed network. Users do not want invalid referential integrity relationships and transaction inconsistency in distributed or local environments. The real-time and recovery properties are met by providing a solution that combines existing methods for local referential constraint enforcement and distributed transaction management. Basically, the RI Constraint manager will interact with the transaction coordinator to utilize a two-phase commitment protocol when performing distributed referential constraint enforcement. This protocol will ensure transaction atomicity at all sites that participate in a distributed constraint enforcement request which will guarantee that all the distributed data is consistent and reliable. In addition, this method does not use any data replication which will make it quite efficient in terms of storage and network communications. The RI Constraint manager will start a nested referential constraint transaction. Since the RI constraint manager will already be executing within a transaction, a direct interface to the Transaction Coordinator is necessary so that this nested constraint transaction can be started. The Transaction Coordinator will then decide if this constraint transaction just needs to be executed locally or if a distributed constraint transaction needs to be started; distributed transactions require a two-phase commit protocol to be utilized. Refer to Fig. 2 for the layout of these two services.

At constraint definition time, the RI Constraint Manager will first determine the location of the related parent table in the specified referential constraint relationship. If this parent table is located at a remote site in our distributed data base network, then this "distributed" resource must be registered with Transaction Services. This resource registry will then be used by the Transaction Coordinator (TC) to determine if a local or distributed transaction is necessary for the requested table operation. Once the table is properly registered, the Constraint Manager will continue with the constraint creation process which involves associating a constraint descriptor with the parent and dependent table. This constraint descriptor identifies the parent and dependent tables, foreign key column(s), primary key column(s), rules, and other information about this

referential constraint relationship. These descriptors are usually chained off of the table's internal control block so that they are easily accessible during constraint enforcement. When both tables are located on the same site the creation process just continues on.

However, if the parent table resides on a remote location, then the constraint manager must ask the TC to start a distributed constraint creation transaction. The TC will then utilize two-phase commit to ensure that the constraint descriptor is either successfully added at each site or that the constraint creation request is aborted at each site. See Fig. 3 for a flow chart of the creation process. Constraint descriptor changes (e.g., state or attribute changes) will be handled in a similar fashion. The RI Constraint Manager will then utilize to Transaction Services to ensure that the same constraint descriptor changes are made to the parent and dependent tables regardless of their location. Any data operation or manipulation by the user to a parent or dependent table causes the RI Constraint Manager to take control. Since RI Constraint enforcement can impact several tables, the Constraint Manager will always request the Transaction Coordinator to start a nested constraint enforcement transaction so that all subsequent enforcement operations are grouped under the same transaction. The Transaction

Referential Integrity Implementation Details and Advantages

A method for providing referential integrity within a database system that provides both superior functionality and ease-of-use is disclosed. This solution also allows for easy referential constraint management without drastically impacting system storage requirements as other methods have done. In addition, this method is also standards compatible with another key database feature, triggers, and can be applied to distributed database systems as well.

An important function within any relational data base system is that of referential integrity. Referential Integrity (RI) ensures data consistency between related columns of two different tables or the same table. The defined and enforced relationship between these related columns is known as a referential constraint. A referential constraint is an assertion that non-null values of a "foreign key" are valid only if they also appear as values of the "primary" key in the related table. To guarantee a data base's referential integrity, the data base system must ensure that non-null foreign key values have a matching primary key value. Although the concept of referential integrity is not new, the RI design disclosed here is unique and advantageous in many aspects. Disclosed are four methods for referential integrity.

1. Constraint Management - Ease of use in managing table constraints with missing pieces: The user can have tables with partial constraints. To define a constraint relationship, the parent table and dependent table must both exist. The system allows the user to define a constraint relationship even when the object (a file member in DB2/400*) that contains the data does not yet exist.

   The mechanism to support this is the system cross reference file. When a constraint is defined, an entry for the constraint will be registered in the cross reference file. Once the parent and dependent table are associated with data objects (members), a dependent access path (index) will be created for the constraint definition, the constraint will be registered in the cross reference file, and the constraint will be placed in the established state.

   Other systems also use indexes for RI constraint definitions. However, most of those systems always create their "own" index for constraint enforcement. This method has the database first look and see if the table has an existing index that can be shared for constraint enforcement before creating an index.

   This provides the end user with a couple of advantages. First, additional storage (DASD) for a separate index is not wasted by the database system in the case where this method is able to share an existing index (which should be fairly common). Second, changes to that table will not suffer the performance degradation associated with maintaining "extra" indexes over a table.

   The system will also automatically attempt to reestablish the constraint relationships when a table (or file) moves to another database (or library) via a copy operation, table rename, or load operation from media. The system cross reference file is again used to deter-

mine which tables and columns participate in a relationship so that the system can redefine or reestablish the constraint relationship when a table is "moved" by the user.

2. Constraint Enforcement (& Trigger Interaction) - Disclosed is a method for ensuring that the enforcement of table constraints and the execution of trigger programs is consistent and predictable.

This method relies on deferring constraint checking until the end of the statement or transaction is necessary to prevent any table/constraint restrictions and to prohibit table constraints and trigger operations from affecting Structured Query Language (SQL) selection results as dictated by the American National Standards Institute (ANSI) SQL-92 standards. An enforcement ordering algorithm and a deferred enforcement mode to ensure that the interaction between table constraints and trigger operations is consistent and predictable.

Some definitions of terms used in this publication will first be discussed.

- Statement - One or more row manipulations due to a single user request. For this paper, a statement is only defined within commitment control.
- Transaction - One or more statements associated with a commitment control boundary.
- Enforcement Cycle - The mechanism for defining the boundaries of either a statement or transaction. Also known as a nested referential constraint transaction.

The execution of a statement by a database user results in the manipulation of a row or rows by the Data Base Management System (DBMS). These row manipulations can be the result of enforcing the table constraints as a result of the user request. For example, referential update cascade rule enforcement may cause dependent records to be updated or trigger programs may cause parent records to be inserted.

The ANSI SQL-92 standards require the evaluation of table constraints to be performed at the end of the statement or transaction. This evaluation deferral must be isolated from the primary transaction. For example, if a user has started a primary commit cycle and has performed a number of updates before issuing a delete causing a delete cascade to occur, an exception causing the delete cascade to get rolled back should not roll back the users original updates.

Evaluation of the constraints is then performed at the end of this enforcement cycle to meet the standards. Thus, the evaluation of constraints is performed after all user initiated row manipulation within the statement have completed, including manipulations caused by trigger programs. The evaluation is done in the following order to eliminate the need to restrict environments of cyclic or connected tables and to consistently and predictably enforce table constraints on files that are trigger enabled. See Fig. 1 for an example of delete connected files.

a. Start enforcement cycle

b. Before Trigger Program execution

c. Restrict RI rules are enforced first - No Action rules differ from Restrict rules in that No action rules are enforced last. Since they are enforced first, Restrict rule enforcement cannot be influenced by other constraints or trigger operations.

d. After Trigger Program Execution.

e. End Enforcement Cycle - The end of the enforcement cycle indicates that table constraint enforcement can now begin. Notice that RI enforcement for all rules other than Restrict will be performed on the after affects of the trigger operations.

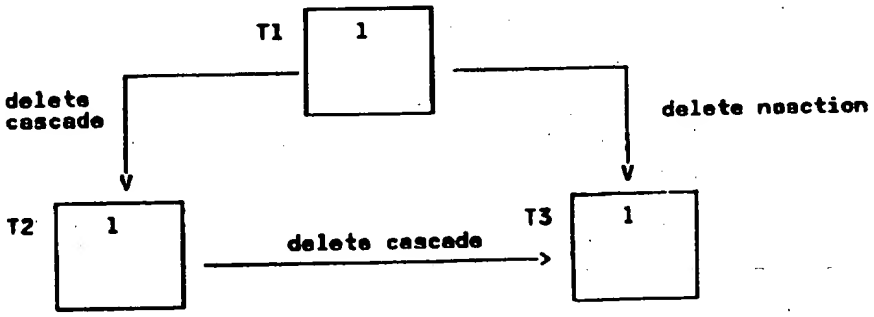
f. All Delete Cascade rules are enforced next - This allows the rows deleted as a result of the Delete Cascade rule to not interfere with the enforcement of the remaining rules.

g. Update Cascade and Delete/Update Set Null and Set Default rules are enforced next.

h. No Action rule enforcement is performed - This enforcement is performed against the results of all other rule enforcement that caused row manipulations. This allows the previous rule row manipulations the chance to clean up any RI violation that otherwise would have caused this transaction to fail.

i. Enforcement of unique table constraints is performed last - Duplicate key exception cannot be raised until the end of a statement. Again, this allows the previously enforced rules the opportunity to clean up this duplicate situation.

This rule enforcement ordering provides the functionality of allowing table constraint enforcement to be performed on the after images of the trigger operations. This would, for example, allow an insert trigger program to be defined over a parent file that would insert the matching dependent record. This could be a users application for populating his files having referential constraints.



If the record in table T1 is deleted, the order in which the constraints is enforced determines the results.

- If the delete noaction is enforced first, the operation results in a constraint violation since a dependent record of 1 exists in table T3.
- If the delete cascades to T2 is enforced first followed by the delete cascade to T3 followed by the enforcement of the delete noaction, then the noaction rule does not result in a violation since the cascade from table T2 has deleted the dependent row in T3.

Figure 1. Example of deleted connected files through multiple paths

If the record in table T1 is deleted, the order in which the constraints is enforced determines the results.

- If the delete noaction is enforced first, the operation results in a constraint violation since a dependent record of 1 exists in table T3.
- If the delete cascades to T2 is enforced first followed by the delete cascade to T3 followed by the enforcement of the delete noaction, then the noaction rule does not result in a violation since the cascade from table T2 has deleted the dependent row in T3.

Since enforcement is delayed until the end of the enforcement cycle, selection results are not impacted by the enforcement during the user statement are not affected. In our PartID example, if all rows with a PartID = '99' were selected for deletion, an update cascade rule adding a new Partid = '99' would not occur until the selection of the original value was complete.

Since the enforcement ordering algorithms ensures that all results are predictable for all constraint environments, there is no need for restrictions. In the delete connected tables example of Fig. 1, since our rule ordering dictates that cascade rules are handled before no action rules, the result is that all rows are deleted. This result is consistent and predictable knowing the order of rule enforcement.

When the user is using statement level constraint deferment, the enforcement cycle is defined to be a nested cycle within the users primary commit cycle. Thus, any exceptions taken during constraint enforcement results in the rollback of the nested cycle, leaving the users primary cycle unaffected.

The rule ordering algorithm implemented in this method defines that delete cascade rules are always enforced before no action rules. Thus, it is predictable and consistent that the result of this operation is that all three tables become empty after the delete of the record in table T1.

3. Distributed Referential Integrity - This methods provides a means for providing RI enforcement across multiple remote server sites when processing distributed transactions.

The key to this method is extending the "local" DB2/400 RI solution with some help from the distributed a distributed database network with some help from a transaction manager. This distributed solution provides real-time referential constraint enforcement and full recovery for any referential transaction; these two properties are a must for ensuring that the data integrity and data consistency exist at ALL sites in a distributed network. Users do not want invalid referential integrity relationships and transaction inconsistency in distributed or local environments.

The real-time and recovery properties are met by providing a solution that combines existing methods for local referential constraint enforcement and distributed transaction management. Basically, the RI Constraint manager will interact with the transaction coordinator to utilize a two-phase commitment protocol when performing distributed referential constraint enforcement. This protocol will ensure transaction atomicity at all sites that participate in a distributed constraint enforcement request which will guarantee that all the distributed data is consistent and reliable. In addition, this method does not use any data replication which will make it quite efficient in terms of storage and network communications.

The RI Constraint manager will start a nested referential constraint transaction. Since the RI constraint manager will already be executing within a transaction, a direct interface to the Transaction Coordinator is necessary so that this nested constraint transaction can be started. The Transaction Coordinator will then decide if this constraint transaction just needs to be executed locally or if a distributed constraint transaction needs to be started; distributed transactions require a two-phase commit protocol to be utilized. Refer to Fig. 2 for the layout of these two services.

At constraint definition time, the RI Constraint Manager will first determine the location of the related parent table in the specified referential constraint relationship. If this

parent table is located at a remote site in our distributed data base network, then this "distributed" resource must be registered with Transaction Services. This resource registry will then be used by the Transaction Coordinator (TC) to determine if a local or distributed transaction is necessary for the requested table operation. Once the table is properly registered, the Constraint Manager will continue with the constraint creation process which involves associating a constraint descriptor with the parent and dependent table. This constraint descriptor identifies the parent and dependent tables, foreign key column(s), primary key column(s), rules, and other information about this referential constraint relationship. These descriptors are usually chained off of the table's internal control block so that they are easily accessible during constraint enforcement. When both tables are located on the same site the creation process just continues on. However, if the parent table resides on a remote location, then the constraint manager must ask the TC to start a distributed constraint creation transaction. The TC will then utilize two-phase commit to ensure that the constraint descriptor is either successfully added at each site or that the constraint creation request is aborted at each site. See Fig. 3 for a flow chart of the creation process.

Constraint descriptor changes (e.g., state or attribute changes) will be handled in a similar fashion. The RI Constraint Manager will then utilize to Transaction Services to ensure that the same constraint descriptor changes are made to the parent and dependent tables regardless of their location.

Any data operation or manipulation by the user to a parent or dependent table causes the RI Constraint Manager to take control. Since RI Constraint enforcement can impact several tables, the Constraint Manager will always request the Transaction Coordinator to start a nested constraint enforcement transaction so that all subsequent enforcement operations are grouped under the same transaction. The Transaction Coordinator will then make the decision on whether the nested constraint enforcement will require just a local transaction or a distributed enforcement request. The RI Constraint Manager will then wait for the Transaction Coordinator to report whether the nested enforcement transaction completed successfully. If the transaction completed successfully, then the data operation is allowed to complete. If any part of the constraint enforcement fails, then the enforcement transaction is rolled back at every site and the user at the originating site is informed of the referential integrity violation. See Fig. 4 for this enforcement process.

4. RI Object Relationship Synchronization - This method ensures that When one or more files in a related set is replaced, as in reloading from a backup copy, that the DBMS ensures the data in the replaced files meets the requirements of the defined relationships like RI relationships.

This solution provides a means in which the database management system determines the synchronization state of the files enabled relationships. It is important to note that this method bypasses the expensive (both Central Processing Unit (CPU) and I/O demands) determination when the relationships are synchronized, thus provides earlier object access and availability with much less system stress.

With this method, a DBMS can replace one or more objects in a set of related objects or even move/replicate the set or a subset of the objects and be able to determine if the objects are synchronized without performing verification, thus providing much quicker object access without requiring nearly as much system resource to make the objects available.

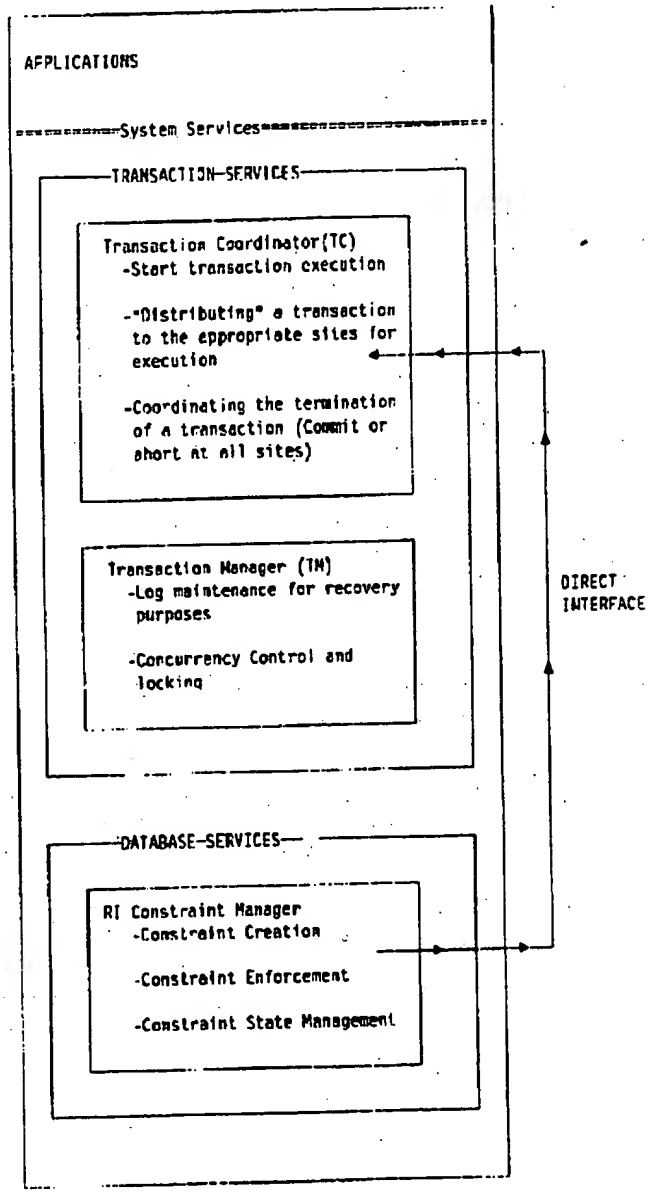


Figure 2. Compute Site X

To better understand the method described, the key terms used in this method are now defined:

- Relationships - A relationship is a rule placed on the objects within the scope of its definition. For example, one can define a relationship in which a particular value cannot reside in file A without also residing in file B. The relationships can either be enabled or disabled. Enabled relationships are enforced as the file's content changes. Disabled relationships are not enforced. The user has control over the Enabled vs Disabled states.
- Enforcement - Enforcement is the act of verifying the modification of a file meets the enabled relationships definition. The modification is not allowed if a violation is found. If a file is replaced or a disabled relationship is enabled, the enforcement entails verifying

each item within the file against all relationships. In this scenario, when violations are found the relationship is put into an invalid state known as Check Pending.

- Check Pending - Check Pending is a state of an enabled relationship. When an established relationship has been violated, the relationship is said to be in the Check Pending state. A file can neither be read nor modified if it contains a Check Pending relationship. Valid is the state the relationship is known to be in if no violations exist. The database management system controls the Check Pending vs Valid transitions.

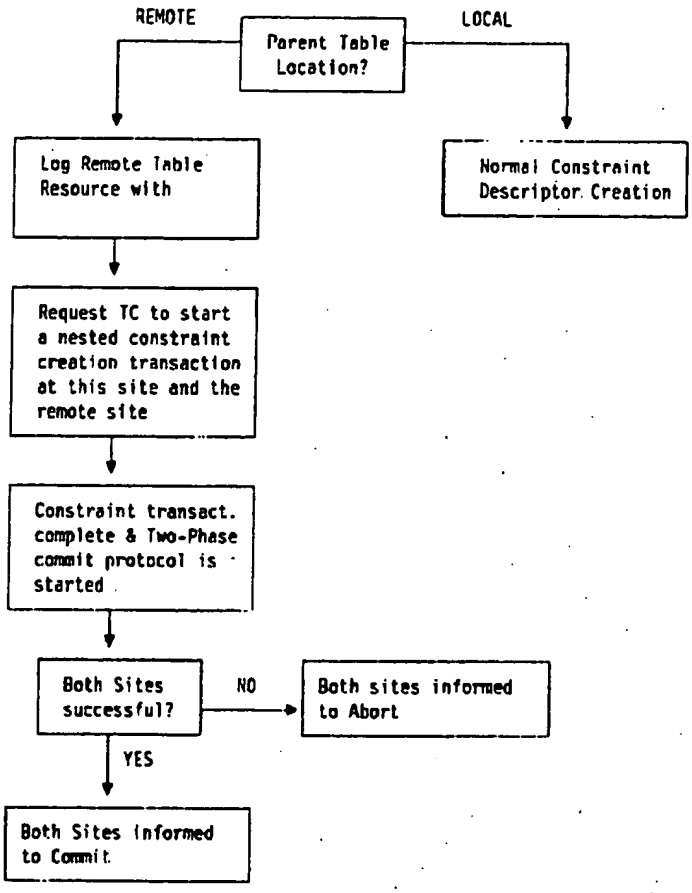


Figure 3. Constraint Creation

In order to solve the synchronization problem the system needs unique relationship identification and data level identification. These two keys will be the building blocks for our method.

Unique relationship identification is necessary to distinguish one relationship from another. That is, we must be able to distinguish any differences among the relationships established before and after the object(s) replacement or relocation. This is critical since one can remove a relationship and add it again with the same name but a different definition. Our relationships will be identified via a name (either user or system defined) and a system generated identifier. This combination will provide unique relationship identification.

Data level identification is necessary to describe the content (state of the data within) the object at any given time. This will be used to determine the synchronization state of the relationship. Data level identification is a modification counter combined with a system identifier. The system identifier is used to distinguish the origin system in which the changes were made since an object may move from one system to another, especially in Client/Server environments. Scope the data level identifier to the file verses each individual relationship. This eliminates the need to visit each relationship when the modification is made (which could degrade the performance of the file modification) while still providing the identification for the reload scenario.

The modification counter need only be incremented if the modification being made affects the enabled relationships. That is, if the modification has no bearing on the relationship definition, the modification counter need not be incremented. This scheme will reduce frequency of counter changes while eliminating the occurrences of object state differences which do not affect the defined relationships.

Since this method scopes our data level identifier to the object level, we need to collect this information during our save processing. That is, for each relationship in file A, we will collect the data level identifier for each object file A is related with. One should note that each relationship which is either disabled or in check pending need not collect the data level identifier. This is appropriate since this relationship state cannot be enabled as a result of this object being loaded. Of course, if one scopes this identifier to the relationship level, this work need not be done.

When an object is loaded onto the system, our load processing will compare the object level identifiers per enabled relationship to determine the relationship state. Process each relationship enabled on the system for a match against the set of relationships which existed at save time. When relationship identifiers and their data level identifiers match, the relationship is moved to the enabled valid state. Otherwise, the relationship is moved to the enabled check pending state.

The following examples describe scenarios that further illustrate the attributes of this method:

Example #1 - Load over existing objects: - Save all files in a single request. Load all files in one request back onto the same system.

Save Operation: Save PF1, PF2 & PF3 in one request.

- PF1's relationship data is collected. It will contain PF1's two relationships with their data level IDs, A/S1-03 and B/S1-03. PF1 is saved to media.
- PF2's relationship data is collected. It will contain PF2's one relationship with its data level ID, A/S1-01. PF2 is saved to media.
- PF3's relationship data is collected. It will contain PF3's one relationship with its data level ID, B/S1-01. PF3 is saved to media.

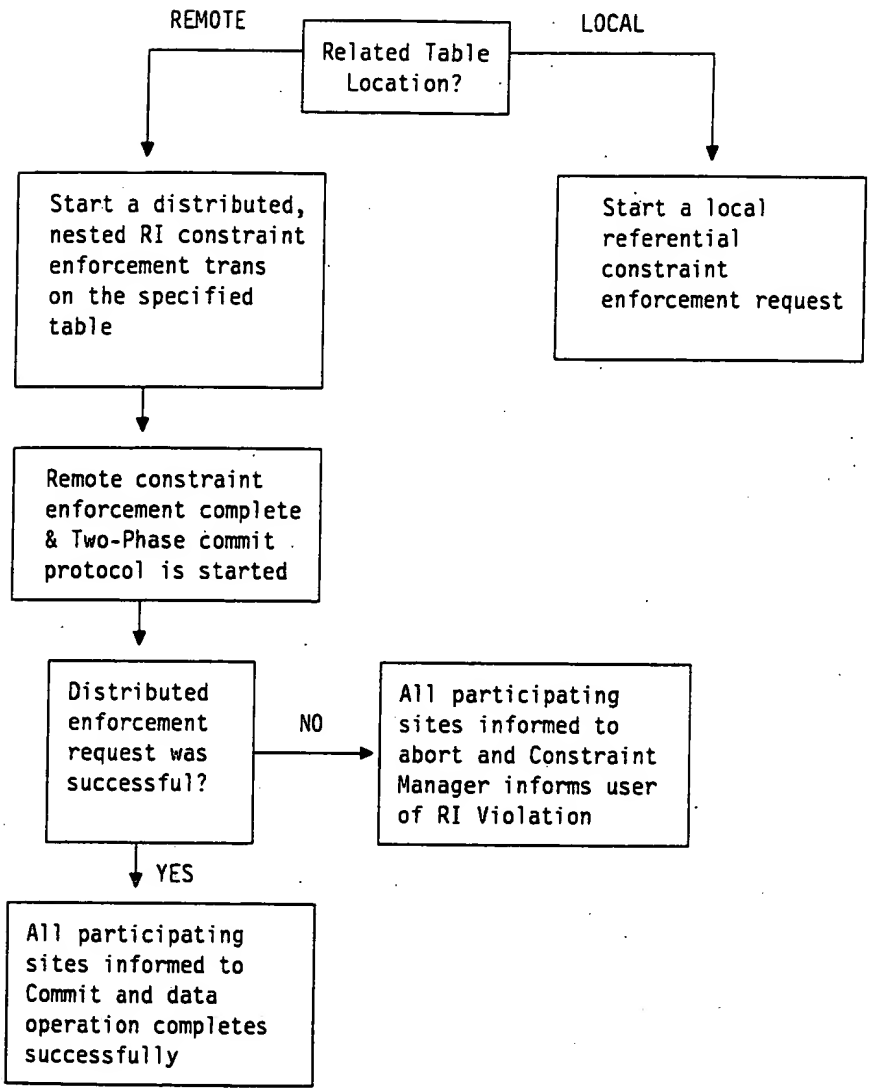


Fig. 4. Constraint Enforcement

Load Operation: Restore PF1, PF2 & PF3 in one request back onto System S1.

- Files exist, no pre-create needed.
- Relationships exist, no pre-define needed.
- PF1 is loaded first, its data level Identifier is loaded and becomes S1-01.
  - Relationship name of A is found on the system. A match (media S1-03 and system S1/03) occurs and the relationship state is set to "valid".
  - Relationship name of B is found on the system. A match (media S1-03 and system S1/03) occurs and the relationship state is set to "valid".
- PF2 is loaded next, its data level Identifier is loaded and becomes S1-03.

  - Relationship name of A is found on the system. A match (media S1-01 and system S1/01) occurs and the relationship state is set to "valid".

- PF3 is loaded next, its data level Identifier is loaded and becomes S1-03.
  - Relationship name of B is found on the system. A match (media S1-01 and system S1/01) occurs and the relationship state is set to "valid".

This method allowed synchronization determination to be made without actually replaying the data against the established rules. Our competitors would have performed the reverification and taken much longer to complete load.

Example #2 - Load where files do not exist: - Save PF1, PF2 & PF3 in one request. Loading all of the files in one request in the following manner yields the same result:

- Delete the files before loading back into the saved from library.
- Load into a new library where the files do not exist.
- Load onto a new system where the files do not exist.

Save Operation: Save PF1, PF2 & PF3 in one request. Same steps as the save operation on the previous example.

Load Operation: The following scenarios yield identical results:

- Delete the files before loading back into the saved from library.
- Load into a new library where the files do not exist.
- Load onto a new system where the files do not exist.

- The operating system pre-creates the files and defines all the enabled relationships which existed at save time. Note the three files will have a new data level identifier, lets say it will be S1/05 if the load is back onto the same system and say S2-0N when going to the new system.

- PF1 is loaded first, its data level Identifier is loaded and becomes S1-01.
  - Relationship name of A is found on the system. A mismatch (media S1-03 and system S1/05 or S2-0N) occurs and the relationship status is set to "check pending".
  - Relationship name of B is found on the system. A mismatch (media S1-03 and system S1/05 or S2-0N) occurs and the relationship status is set to "check pending".

- PF2 is loaded next, its data level Identifier is loaded and becomes S1-03.
  - Relationship name of A is found on the system. A match (media S1-01 and system S1/01) occurs and the relationship status is set to "valid".

- PF3 is loaded next, its data level Identifier is loaded and becomes S1-03.
  - Relationship name of B is found on the system. A match (media S1-01 and system S1/01) occurs and the relationship status is set to "valid".

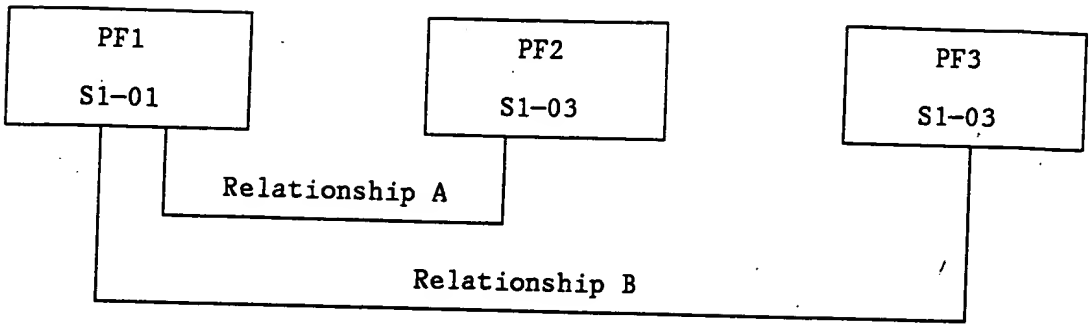


Fig. 5. Relationship network #1, all in library A.

Although this idea provides a richer Referential Integrity function, it could be used in any environment that introduces a relationship among objects (Fig. 5).

* Trademark of IBM Corp.